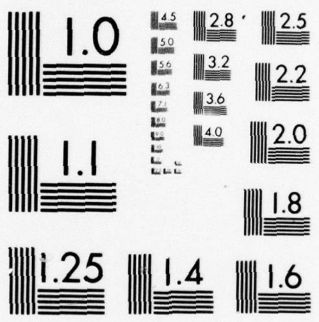
AD-A067 332   MARYLAND UNIV COLLEGE PARK COMPUTER SCIENCE CENTER   F/G 12/2
LOCAL RECONFIGURATION OF NETWORKS OF PROCESSORS.(U)
FEB 79 A WU, A ROSENFELD   AFOSR-77-3271
UNCLASSIFIED   TR-730   AFOSR-TR-79-0443   NL

1 OF 1
AD
A067332

END
DATE
FILMED
6 - 79
DDC


1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

UNIVERSITY OF MARYLAND

COMPUTER SCIENCE CENTER

COLLEGE PARK, MARYLAND
20742

18 AFOSR 19 TR-79-0443

12

14 TR-730
15 AFOSR-77-3271

11 February, 1979

12 42p.

6 LOCAL RECONFIGURATION OF NETWORKS OF PROCESSORS.

10 Angela Wu
Dept. of Mathematics
University of Maryland
Baltimore, MD 21228

Azriel Rosenfeld
Computer Science Center
University of Maryland
College Park, MD 20742

9 Technical rept.,

ABSTRACT

This paper studies a "local" approach to reconfiguration of networks of processors, in which new connections are created only between pairs of processors that have a common neighbor. Algorithms for transforming strings into cycles, trees, arrays, hypercubes, cliques, stars and wheels, and vice versa are presented. The generation of all possible configurations of a given set of processors is also discussed.

The support of the U.S. Air Force Office of Scientific Research under Grant AFOSR-77-3271 is gratefully acknowledged, as is the help of Kathryn Riley in preparing this paper.

1. Introduction

1.1 Networks of Processors

A network of processors is often represented by a connected graph where a processor is placed at each node and an arc between two nodes stands for a direct connection between the processors at the nodes. Whenever no confusion can arise, we will use "nodes" and "processors," "arcs" and "connections" interchangeably. The degrees of the nodes are usually bounded since the number of processors one can be directly connected to is physically limited. This restriction becomes significant if one wants to deal with very large networks.

In the study of networks of processors, there is growing interest in self-reconfiguration of the network [1-4]. Reconfiguration may be required because, for example, (1) the processor at a node is malfunctioning, (2) a communication line (arc) between two nodes is broken, (3) a new node becomes available, or (4) different computational tasks can be performed more efficiently using different configurations. This suggests the design of self-reconfigurable networks of processors having a fixed permanent connection structure, and also having the capability of generating temporary structures without disturbing the permanent structure. The permanent structure is desirable in order to insure that connectedness is always preserved no matter what the temporary structure may be. It is desirable that the permanent structure be multi-connected so that it can tolerated, detect and heal "faults" such as malfunctioning nodes or broken arcs in the system. The temporary structures should be easy to obtain from the permanent structure and be able to perform the desired tasks efficiently.

1.2. Local reconfiguration

In many network systems, the interconnections of processors are specified by a switching network [5,6]. However, this approach has several disadvantages: (a) If the control of the switching network malfunctions, the interconnections are destroyed. (b) A switch cannot handle a large number of signals simultaneously; thus the possibility of a high degree of parallelism is lost. (c) The many levels of switches required may cause delays; for example, in [6] it is shown that the switching network of a hypercube of 2^n nodes has n levels and needs $n2^{n-1}$ switches.

This paper proposes a "local" approach to reconfiguration, in which reconfiguration is achieved by repeated local arc creation and deletion. (We assume, for the moment, that the node set does not change.) Arc deletion is evidently a local process; the end nodes of the arcs are marked to signify that the arc is considered to be disconnected. (We assume here that the arcs at each node have distinct identifiers, so the mark can indicate which of the arcs is disconnected.) The addition of an arc by local processing is more difficult, since the two nodes to be connected may be very far away from each other. Furthermore, many nodes may all want to be joined with the same node, so that it may not be possible to maintain boundedness of degree.

If we want to add arcs by local processing, we must first require that both end nodes agree that they are to be connected. Suppose nodes m and n are at distance 2 apart with a common neighbor

k. If node m wants to join to n, then it signals to node k which checks with node n. When node k receives a positive repl from node n, it signals nodes m and n, and a new arc is established between m and n.

Now suppose that node m wants to join to a node n which is at distance p away. Let $m,k_1,k_2,\ldots,k_p=n$ be a path from m to n Then m can first connect to k_2 using k_1 as the common neighbor. Next, m connects to k_3 using k_2 as the common neighbor and disconnects the temporary arc to k_2 at the same time. Continuing in this way, m is finally joined to n. This process is illustrated in Figure 1.

Note that if a given bound d on the degrees of nodes is to be preserved in the new configuration, a new arc should be created only when the nodes involved (m and n) have degrees less than d. However, there is no guarantee that the nodes along the path from m to n all have degrees less than d; hence intermediate degrees higher than d may have to be allowed. This is not serious, since the intermediate degrees are still bounded and known. In case there are multiple requests for new arcs to be created, they will be processed simultaneously provided the degree restrictions (if any) are satisfied; otherwise the processor must select the request to be filled at each step.

This paper discusses local reconfiguration in the context of a formal model for networks of processors, called a <u>cellular d-graph automaton</u> [9-15]. In this model, identical processors are interconnected uniformly into a graph. Each processor is a node of the graph. The network is "autonomous" in the sense that the next state of a processor (which operates in discrete time steps) depends only on its own state and the state of its neighbors in the graph. Thus communication between two nodes at distance k apart takes k time steps; the message is relayed by the nodes along a path connecting the two given nodes. A bound d is imposed on the degree of each node, since it is not possible to connect a processor directly to a very large number of processors. In this case we may assume that the neighbors are all distinguished by having the arc ends numbered 1 through d. The advantages of using cellular systems as networks of processors are discussed in [7,8]. In [11-15] the cellular d-graph automata considered have a distinguished processor which is regarded as specially marked.

Cellular d-graph automata are studied in [9-15]. In these papers, the automata are assumed to have fixed graph structure, whereas in the present paper they are allowed to locally reconfigure themselves. It is an open question whether this increases their power as graph language acceptors.

1.3. Some specific configurations

In this paper we investigate local reconfiguration algorithms for transforming strings, cycles, stars, wheels, trees, arrays, hypercubes and cliques into one another. Specifically, we show how to transform a string into each of the other types and vice versa. Thus one can get from any type to any other using a string as an intermediate configuration. Algorithms for transforming one type directly into another could also be given, but will not be presented here. Note that stars, wheels, hypercubes and cliques do not have bounded degree.

Any of these types of graphs might be appropriate as a temporary configuration, depending on the tasks to be performed. In particular, trees have the fewest possible arcs for a given number of nodes (since we require connectedness), and also have low diameter, which is advantageous for fast interprocessor communication. On the other hand, a tree disconnects if an arc or a nonleaf node fails, so that it would not be a good permanent structure from the standpoint of fault tolerance. A cycle, being biconnected, is a better permanent structure; and if bounded degree is not a problem, a clique might be the ideal permanent structure.

Sections 2-7 present transformations between strings and cycles, trees, arrays, hypercubes, cliques, and stars or wheels, respectively. Section 8 discusses how to generate all possible temporary configurations systematically; it assumes that each node can count up to O(log n), where n is the number of nodes.

2. String↔Cycle

Ring (or loop) networks whose underlying configuration is a cycle are of considerable interest in the design of local computer networks [5]. In this section we will show how to change a string into a cycle in diameter time, and also discuss reconfiguration when extra processors are available.

Given any string, it is easy for a network of processors with a distinguished node D to make one end of the string the distinguished end as follows: If D is one of the ends, then it is the distinguished end. Otherwise D has two neighbors and it sends the lower numbered neighbor a signal which propagates to its other neighbor. Eventually this signal reaches an end node and this end node is the distinguished end. We will refer to this distinguished end as the left end node. It is easy to define a consistent arc end numbering such that 1 leads to the right neighbor and 2 leads to the left neighbor. All these steps can be done in diameter time.

Once the left end node m of a string of length greater than 2 is identified, it is very easy for m to join itself to the other end node n in linear time, since the path from m is fixed. The string thus becomes a cycle.

From this it is easy to see that a cycle can detect and tolerate one fault, because it then becomes a string and can heal itself, i.e., reconnect into a cycle.

It is also very simple to turn a cycle into a string, since this simply requires the deletion of one arc, say the lower numbered arc of the distinguished node.

If an additional processor becomes available to be attached to one of the processors on the cycle, this processor can act as the common neighbor to join the new processor to one of its neighbors and disconnect its old link to this neighbor. The new configuration is then a cycle including the new processor. See Figure 2.

3. String$\leftrightarrow$Tree

Trees provide good control of message transmission since each node has only one father node which may be specified as the sender or receiver of particular messages. A balanced k-ary tree has degree k+1 and the distance between any two nodes is at most $2*\lceil \log_k(\text{number of nodes})\rceil$. Hence a balanced k-ary tree is a good temporary configuration.

3.1. String$\leftrightarrow$Balanced Binary Tree

A balanced binary tree is a binary tree such that at each node, the numbers of nodes in its left subtree and its right subtree differ by at most 1. For any d-graph having more than five nodes to be a balanced binary tree, d must be at least 3.

In this section we will show that for any string, the arcs can be reconnected into a balanced binary tree in diameter (of string) time. The basic idea is to identify the midpoint m of a string as the root of the tree and then reconnect it with the roots of the left and right subtrees (if any), which are the midpoints of the substrings on the two sides of m. This process is then repeated.

To identify the midpoints, one end of the string sends out two signals, U at unit speed and T at 1/3 speed. When U reaches the other end of the string, it bounces back. Signals U and T will meet at the midpoint, i.e., a node at distance $\lceil \ell/2\rceil-1$ from the node sending out the signals T, U. The midpoint is marked and both of its neighbors send out signals T and U to find the midpoints of the corresponding substrings. Moreover, the signal T can also serve to guide the intermediate temporary links so that when the new midpoint is identified, it is connected to its father node in the tree. When a just identified midpoint is connected to a new node, its link to the common neighbor is disconnected. The process of finding midpoints and reconnecting stops when the substring has length 1. Figure 3 illustrates the process.

It takes $3\ell/2$ steps to identify the midpoint of a string, and then both sides can identify their own midpoints simultaneously. Since $\frac{3}{2}\ell + \frac{3}{2}\frac{\ell}{2} + \frac{3}{2}\frac{\ell}{2^2} + \ldots + \frac{3}{2} \cdot \frac{\ell}{2^{\log \ell}} < 3\ell$, a string can be reconnected into a balanced binary tree in O(diameter) (of string) time.

3.2. Binary Tree→String

Any node having fewer than three neighbors in a binary tree can serve as the root. The distinguished node sends out a signal to find such a node; the first one (ties broken by choosing the lowest numbered one) to return a "found" message will be designated as the root. Once the root node is determined, it sends a signal to implicitly renumber the arc ends consistently, using 1 to denote the arc leading to the father node, and 2 and 3 to denote the arcs leading to the sons. We will think of the second neighbor as the left son and the third neighbor as the right son. From the new arc end numbering, a node is able to tell if it is the left or right son of its father. Now the root creates a signal A which propagates down to its descendants. When a node n receives signal A, and it is the left (right) son of its father f, then n knows f wants to join to n's rightmost (leftmost) descendant, if any. The reconnection process starts and f is disconnected from n and joined to n's right (left) son k. Now node k has an extra temporary arc numbered 4, and k knows node f wants to join to its right (left) son. This process continues until f is joined to a node m with no right (left) son. Figure 4 shows an example of this transformation. Note that the transmission of signal A continues while f tries to establish a connection with its rightmost (leftmost) descendant. Upon completion, the string gives an inorder traversal of the binary tree. If the tree was obtained by the method of Section 3.1 using the midpoint of the string as the root, then the above will produce the original string.

The time it takes to change a binary tree into a string is proportional to the height of the tree. It depends on the chosen root. However, it is always no larger than the diameter of the graph.

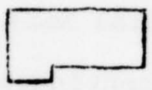
3.3. String$\leftrightarrow$k-ary tree

Given any integer $k \geq 2$, it is straightforward for a network of processors, in O(diameter) time, to mark k nodes, n_0, $n_1, \ldots, n_{k-1}$ of a string such that n_0 is the left end node of the string, and such that the number of nodes between n_i and n_{i+1} (including n_i but not n_{i+1}) for $0 \leq i < k-1$ and the number of nodes from n_{i-1} (inclusive) to the right end of the string differ by at most 1. The method of changing a string into a k-ary tree is analogous to that for the binary tree in Section 3.1. The left end node of the string is used as the root. The rest of the string is divided into k (almost) equal length substrings by the marked nodes $n_0, n_1, \ldots, n_{k-1}$ as above, where n_0 is the neighbor of the left end (root) node. Nodes $n_0, \ldots, n_{k-1}$ become the sons of the left end node and each n_i $(0 < i < k)$ is disconnected from its left neighbor. The nodes between n_i and n_{i+1} not including n_i are then divided into k substrings whose leftmost nodes are the sons of n_i. This is repeated until the substrings have length less than or equal to k; then each node is a son of n_i and the nodes are disconnected from each other. Since the divisions of the substrings are independent of each other, they can be done simultaneously and the construction of the k-ary tree can be achieved in O(diameter) (of string) time.

To change a k-ary tree into a string, as in Section 3.2, the root is determined and the arc ends are renumbered consistently so that a node can always tell which neighbor is its father and which is its ith son. In case a node has less than k sons, they are numbered 1 to j for some $j<k$. The root node creates a signal A which propagates down to its descendants. When a node n receives signal A and it is the ith son of its father f for some $i>1$ then n first joins to its i-1st brother m, and disconnects its link to f. In subsequent steps, n joins to the highest numbered son of m, and disconnects from m. This is repeated until n is joined to a leaf. In the resulting string, the root is the left end node. A node's right neighbor is its first son if it is not a leaf. Figure 5 shows a 3-ary tree and its resulting string. The time it takes to change a k-ary tree into a string is proportional to the height of the tree, i.e. O(diameter) (of tree) time.

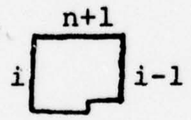
4. String$\leftrightarrow$array

Given any string such that its t-th node m from the left end has a special mark R, it is easy to design a network which marks the i*t-th nodes for $1<i$ and $i*t \leq$ length of string. As shown in Figure 6, the left end node sends out two signals, α at unit speed and γ at 1/2 speed; node m also sends out a signal β at 1/2 speed. When α meets β, α has passed 2t nodes and β has passed t nodes, so that they meet at a node m_1 at distance t from node m, and the special mark R is made on node m_1. Signal α disappears and signal β continues. When signal α meets β at node m_1, signal γ also reaches node m. Whenever signal γ reaches a specially marked node, a new signal α travelling at unit speed is created. After 2*length (string) steps, every t-th node of the string is marked with R, the rightmost node knows whether the length of the string is a multiple of t, and a message is sent back to the leftmost node.

All the nodes marked with R can be considered as the right border nodes of a rectangular array of which each row has t nodes. To make the string into a rectangular array, every node simply has to connect itself to the node at distance t to its right. The left end node initiates a signal δ at 1/2 speed to be propagated to the right. When a node a receives signal δ, it starts to try to join itself to a node b at distance t away. Here b will be the first node past an R node that still has only two neighbors, since the arc from node a reaches node b before any arcs from the nodes to the right of a because δ is transmitted from left to right. After an R node receives δ and is temporarily connected to a node at distance 2 to its right, the R node is disconnected from its right neighbor. If an intermediate temporary link reaches the right end node then the temporary arc is eliminated. Figure 7 shows the first few steps of the process. If the length of the string is a multiple of t then the resulting structure is a rectangular array; otherwise, it is an almost rectangular array of the form  .

. The above shows how to change a string into a rectangular array in linear time when one of its dimensions is given. We next show how to change a string into a square array, again in linear time.

We first show that given any string, say of length r, then in time O(r) the network of processors can place a mark R at the pth node from the leftmost node, where $p=\lceil\sqrt{r}\rceil$. The left end node first sends a signal to mark the second node N and the fourth node S. When a node is marked S, it is the n^2th node from the left for some n and the distance between the nodes marked N and S is n. The node marked N sends out two signals, α at unit speed and β at 2/3 speed. The node marked S sends out signals γ at 1/3 speed and δ at 2/3 speed as shown in Figure 8. β and γ reach the same node after 3n steps, when β has travelled distance 2n and γ distance n. α and δ also meet after 3n steps when α has travelled distance 3n and δ distance 2n. At the next step, the node where β and γ meet is marked N and the left neighbor of the node where α and δ meet is marked S. Then N and S send out the signals again and repeat the process. Note that the distance between the two S nodes is 2n+1, and that between the two N nodes is 2n, so that the distance between the new N and the new S nodes is n+1. By the relation $(n+1)^2 = n^2+2n+1$, the new S node is the $(n+1)^2$th node from the left. The process stops when the signals reach the right end of the string. Clearly the rightmost node can tell if the length of the string is a perfect square and the distance between the last pair of nodes N and S = $\lfloor\sqrt{\text{length of string}}\rfloor$. It is then straightforward to transmit this to the left end of the string and mark the $\lceil\sqrt{\text{length of string}}\rceil$th node with R. Now using the same method as for rectangular arrays, the string can be changed into a square array or



where i=n or n+1.

Since in each case the rightmost node knows whether the length of the string is a multiple of the given dimension or a perfect square, if a complete rectangular or square array is desired, then the process continues only when the rightmost node allows it to do so.

To change an (almost) rectangular array into a string, the network of processors first identifies a convex corner node as the root and then builds a spanning tree of the array. The spanning tree is binary, and using the method in Section 3.2, a string can be obtained from it. The time for all these operations is proportional to the diameter of the array.

5. String$\leftrightarrow$hypercube

In each of the structures considered above, the degrees of the nodes are always bounded regardless of the number of nodes. In this section, we will study changing strings into structures whose degree depends on this number.

A hypercube of degree n has 2^n nodes. We now describe a network of processors that can tell whether the length of the string is a power of 2 in real time. When the answer is affirmative, it can proceed to reconnect the nodes into a hypercube. Suppose the length of the string is 2^n and the nodes are numbered 1 to 2^n from left to right. Node 2^{n-1}, being the midpoint of the string, can be identified in linear time. When this has been done, the network connects node 2^{n-1} to node 2^n, node $2^{n-1}-1$ to node $2^n-1,\ldots,$ node 1 to node $2^{n-1}+1$. First of all, a new arc joins node 2^{n-1} to node $2^{n-1}+2$. In the next step, this new arc is eliminated, a new arc joins node 2^{n-1} to node $2^{n-1}+3$, and a new arc joins node $2^{n-1}-1$ to node $2^{n-1}+1$. Proceeding in this way, each node k stops and joins to a node m as far to the right as possible, such that m is not already joined to any node to the right of k. After 1 is joined to $2^{n-1}+1$, nodes 2^{n-1} and $2^{n-1}+1$ become boundary nodes, and the connection between them is broken. Now, the above can be repeated for each substring of length 2^{n-1} simultaneously. We continue subdividing and reconnecting, allowing signals to be transmitted through the unbroken original arcs only, until the strings have length 2. Figure 9 indicates why this gives a hypercube.

The time it takes to get a hypercube from a string is $C \cdot 2^n + C \cdot 2^{n-1} + C \cdot 2^{n-2} + \cdots + C \cdot 1 = \frac{C(2^{n+1}-1)}{2-1} = 2C \cdot 2^n$ for some constant C, i.e. = O(diameter(string)) time.

Given a hypercube of degree n, a breadth first spanning tree can be constructed in O(diameter)=O(n) time; it is an n-ary tree of height n. Using the result of Section 3.3, this tree is in turn changed into a string in O(n) time.

6. String$\leftrightarrow$Clique

A clique is a completely connected graph. It has the advantages that it remains a clique even when faults occur. It also has the disadvantage that each processor in a clique of size ℓ needs ℓ-1 connections.

Given a string, we showed in Section 2 that a network of processors can make one end of the string the distinguished left end and define a consistent arc end numbering such that 1 leads to the right neighbor and 2 leads to the left neighbor in diameter time. Then the nodes having a common neighbor and not already neighbors of each other will be joined. Some of the nodes at distance more than 3 apart originally have a common neighbor because of the new arcs. Thus in the subsequent steps, these nodes are joined. Continuing in this way, after O(diameter) steps all the nodes are joined to each other and the connections define a clique. In order to have good control of the addition of new arcs, we require that a common neighbor signal the joining of two nodes only if one of the nodes is its left neighbor in the original string. This avoids a node having to process an unbounded number of signals. Figure 10 shows an example of changing a string into a clique.

Given a clique, the neighbors of the distinguished node D know which neighbor of D they are. D sends signals to the neighbors to protect the connections from the ith neighbor to the i+1st neighbor ($1 \le i \le d$). All unprotected arcs except D's first arc are disconnected and the connections define a string.

7. String↔Star and Wheel

In a star graph, all the communications between processors involve the center processor, which is sometimes considered to be the control or switch center.

Figure 11a shows the process of a string changing into a star, with the left end node becoming the center node of the star, in O(diameter)(of the string) time. In each step the left end node is joined to a node n at distance 2 away and node n is disconnected from the common neighbor. When the left end node is joined to the right end node, a star is formed. The degree of the center node is one less than the number of nodes.

Changing a string into a wheel in O(diameter)(of string) time is similar to changing it into a star except that the nodes are not disconnected from the common neighbor, and after the left end node is joined to the right end node, the left end node serves as the common neighbor to join its original neighbor (in the string) to the right end node. This is illustrated in Figure 11b.

A star changes into a string by first connecting the ith neighbor to the i+1st neighbor of the center node C and then disconnecting C from every node but one, say the first, neighbor. A wheel changes into a string by having the center node signal its lowest neighbor to disconnect from one of its neighbors and then disconnect the center node from all the neighbors except the lowest numbered one.

8. Generating all possible configurations

In this section we discuss how a network of m processors can systematically change itself into all possible connected configurations. Of course the permanent structure remains fixed; only the temporary reconfigurations are changed.

Given any network of processors, a cellular d-graph automaton with a distinguished node can order the nodes and connect them into a string according to this ordering [12]. Suppose that d, the maximum number of nodes a node can be connected to, is greater than or equal to m, the number of nodes in the network. Using the method of Section 6, the string can be reconfigured into a clique. Clearly each processor can store an m-1 bit binary number. Starting with the first node in the ordering, each processor systematically generates the m-1 bit binary numbers from 1 to $2^{m-1}-1$. The number at a node is used to select a set of arcs at the node; namely, the ith bit being 1 indicates that the ith arc at the node is chosen. When a number j is generated at node n, if j is consistent with the numbers at all the nodes preceding n in the node ordering, then the node immediately following n generates its number; otherwise, node n generates the number j+1 and again checks with the nodes preceding it. When the last node successfully generates a number and selects a set of connections accordingly, a new graph is formed. The connectedness of this new graph is easily be checked by sending signals through the new graph and through the connected permanent structure. If it is connected, then it is a reconfiguration. Otherwise, the next graph is generated. This is done by adding 1 to the number in the last node; if this number is $2^{m-1}-1$, 1 is added to the number in the second to the last node, and the last node starts again with the number 1; and so on. In this way, all the possible reconfigurations will be generated.

In the above construction, we assume that a processor can be joined to all the other processors simultaneously. This is usually not practical unless the number of processors is small. A more realistic assumption is that each processor can be connected to a maximum of d processors, and each processor can store d+1 binary numbers of $\lceil \log n \rceil$ bits. In this case, each processor can systematically generate d numbers of $\lceil \log n \rceil$ bits each. Of the d numbers, up to d-1 may be 0's and the rest are all distinct. A non-zero number i at a node indicates that the node is connected to the ith node to its right in the ordering, with the last (rightmost) node joined to the first (leftmost) node. When a nonzero number j is generated at a node n, a copy of j is made and is decremented by 1 at each step until it is zero; for each decrease, the node k connected to n serves as the common neighbor to join n to k's right neighbor in the string, while initially n is considered to be connected to itself. After j steps, the copied number becomes zero and n is joined to a node at distance j to its right. This can be done for each non-zero number at a node. Again the numbers at each node are generated systematically from 1 to n starting with the left end node. Consistency and connectivity can be checked as before. Therefore, all the reconfigurations of the network can be generated.

If the processors have memory of bounded size and the number m of processors is very large, it is not clear whether a network of m processors, allowing at most degree d, can change itself into all possible configurations. One reason to suspect a negative answer is that an m-node graph of degree d has up to dm/2 arcs, which seem to require O(n log n) total memory to specify.

9. Concluding remarks

This paper has studied local reconfiguration of networks of processors. It suggests that a reconfigurable network should have a permanent structure for connectivity and fault tolerance, and temporary structures which are reconfigurations of the permanent structure for performing specific computational tasks efficiently. Local reconfiguration algorithms were presented for transforming strings into and from cycles, trees, arrays, cliques, hypercubes, stars and wheels. As a corollary, any of these can be reconfigured into any other using a string as an intermediate configuration. Finally, it was shown that if each node can store numbers of length $O(\log n)$ where n is the number of nodes, then a reconfigurable network can systematically generate all possible temporary configurations.

References

1. G. J. Lipovski, On a Varistructured Array of Microprocessors, IEEE Transactions on Computers, Vol. C-26, 1977, 125-137.

2. Y. Parker and M. Bozyigit, Variable Topology Multicomputer, Proc. of 2nd Euromicro Symposium on Microprocessing and Microprogramming, Venice, 1976, 141-149.

3. A. D. Friedman and F. Saheban, A Survey and Methodology of Reconfigurable Multi-Module Systems, Proc. of COMPSAC 78, 1978, 780-796.

4. R. G. Arnold and E. W. Page, A Hierarchical Restructurable Multi-Microprocessor Architecture, Proc. 3rd Symposium on Computer Architecture, 1976, 40-45.

5. G. A. Anderson and E. D. Jensen, Computer Interconnection Structures: Taxonomy, Characterstics and Examples, ACM Computing Surveys Vol. 7, 1975, 197-213.

6. M. C. Pease, The Indirect Binary n-cube Microprocessor Array, IEEE Transactions on Computers, Vol. 26, 1977, 458-473.

7. S. Ullman, Relaxation and Constraint Optimization by Local Processors, Computer Graphics and Image Processing, Vol. 10, 1979 (to appear).

8. F. L. Van Scoy, Cellular Automaton Systems: Parallel Processing Systems for the Future, Proc. 17th NBS Annual Technical Symposium, 1978, 27-32.

9. P. Rosenstiehl, J. R. Fiksel, and A. Holliger, Intelligent Graphs: Networks of Finite Automata Capable of Solving Graph Problems. In R. C. Read, ed., Graph Theory and Computing, Academic Press, New York, 1972, 219-265.

10. J. R. Jump and J. S. Kirtane, On the Interconnection Structure of Cellular Networks, Information and Control 24, 1974, 74-91.

11. A. Wu and A. Rosenfeld, Cellular Graph Automata I: Basic Concepts, to be published in Information and Control.

12. A. Wu and A. Rosenfeld, Cellular Graph Automata II: Property Measurement, to be published in Information and Control.

13. A. Wu and A. Rosenfeld, Cellular Graph Automata III: Graph and Subgraph Isomorphism, to be published in Information and Control.

14. A. Wu and A. Rosenfeld, Cellular Graph Automata IV: Graph Structure Recognition, to be published in Information and Control.

15. A. Wu and A. Rosenfeld, Cellular Graph Acceptors V: Closure Properties of Cellular d-graph Languages, to be published in Information and Control.

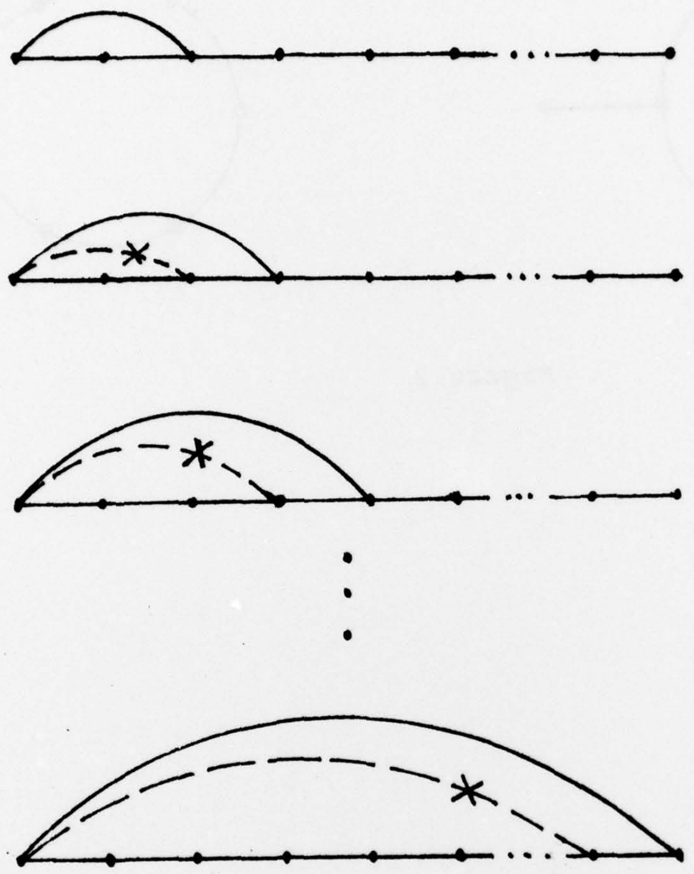


Figure 1

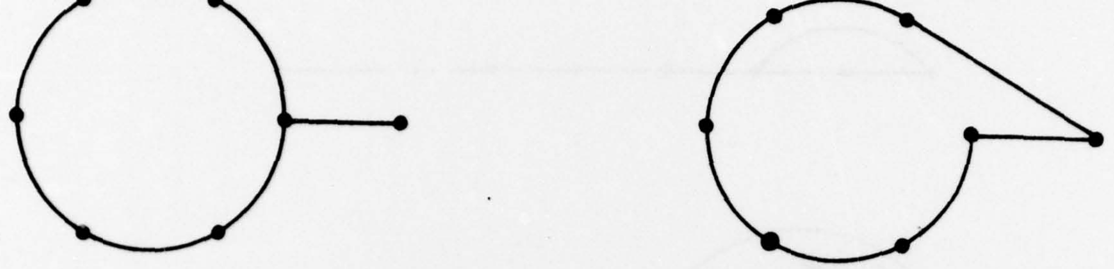


Figure 2

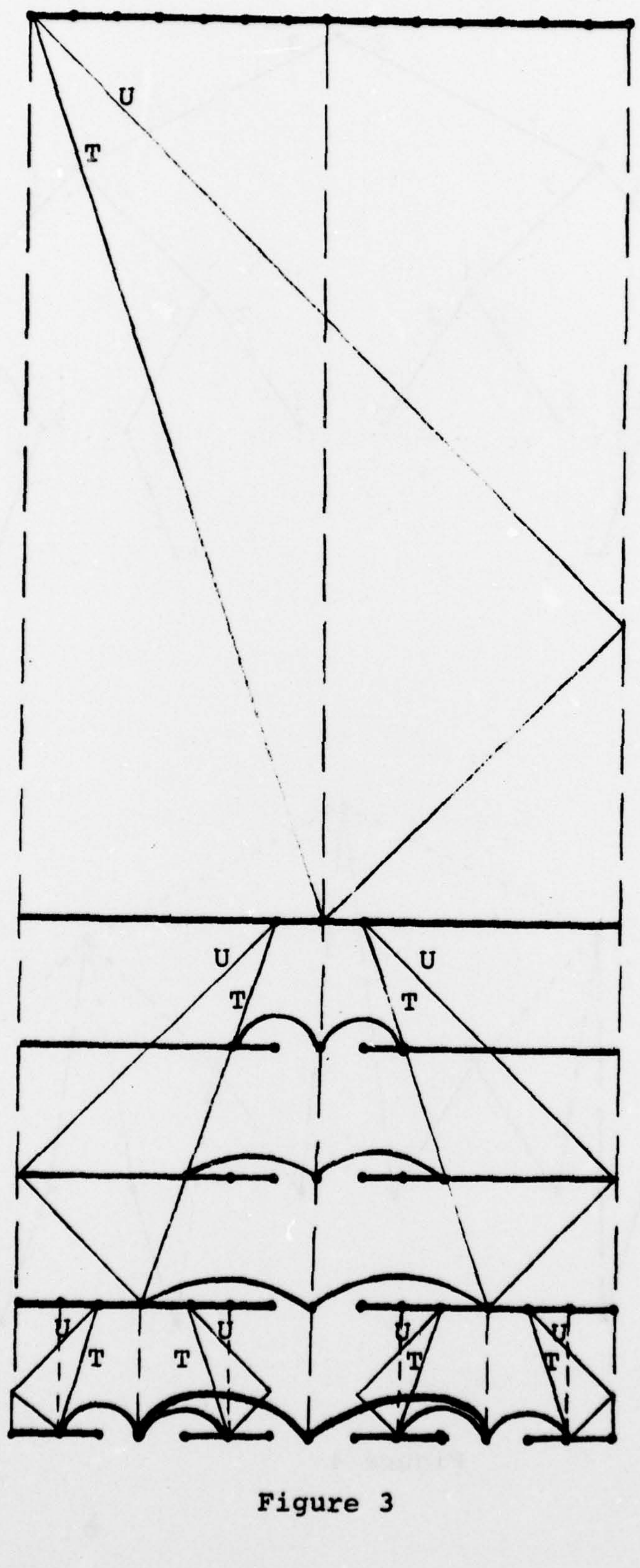


Figure 3

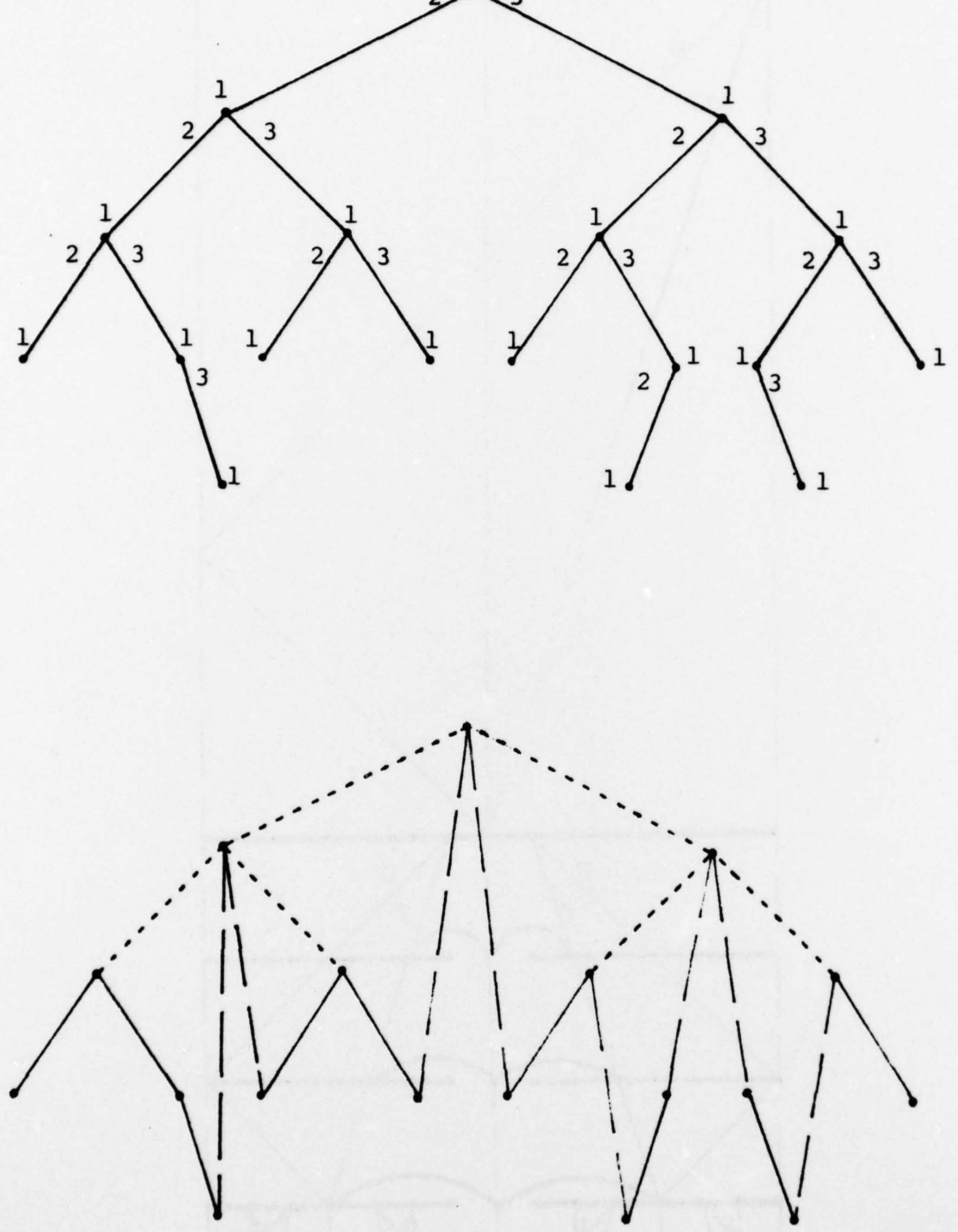
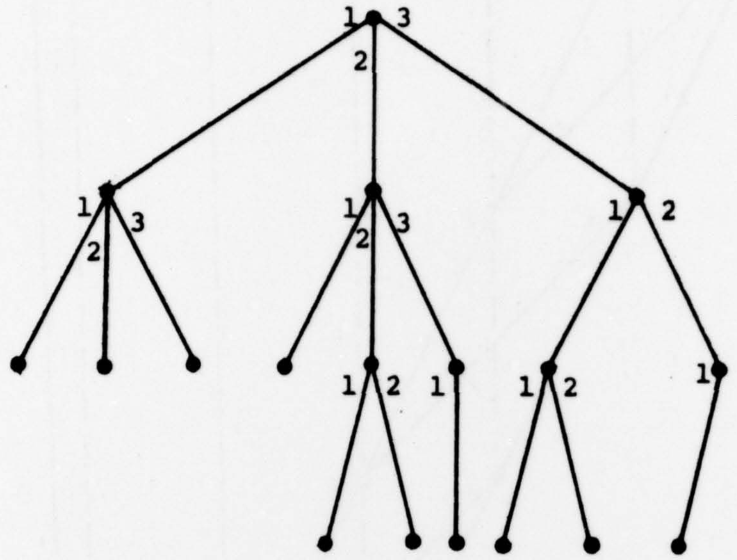


Figure 4



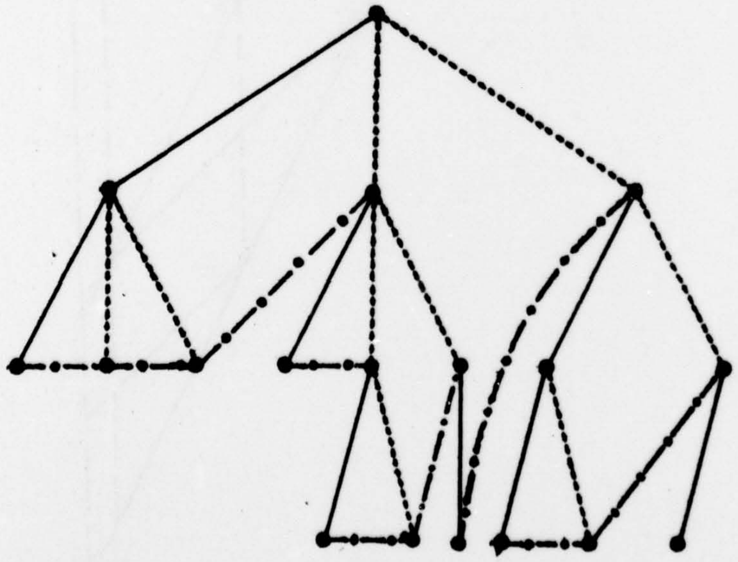


Figure 5

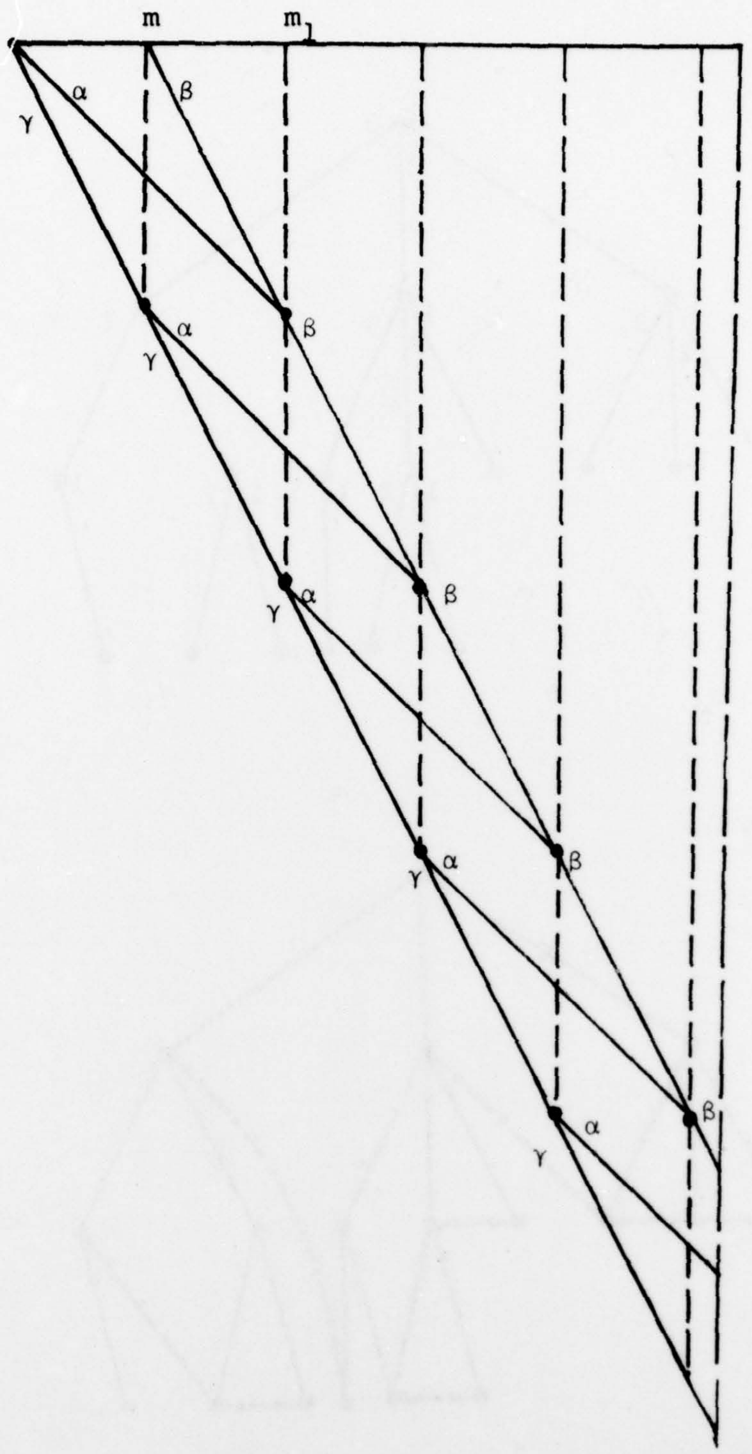


Figure 6

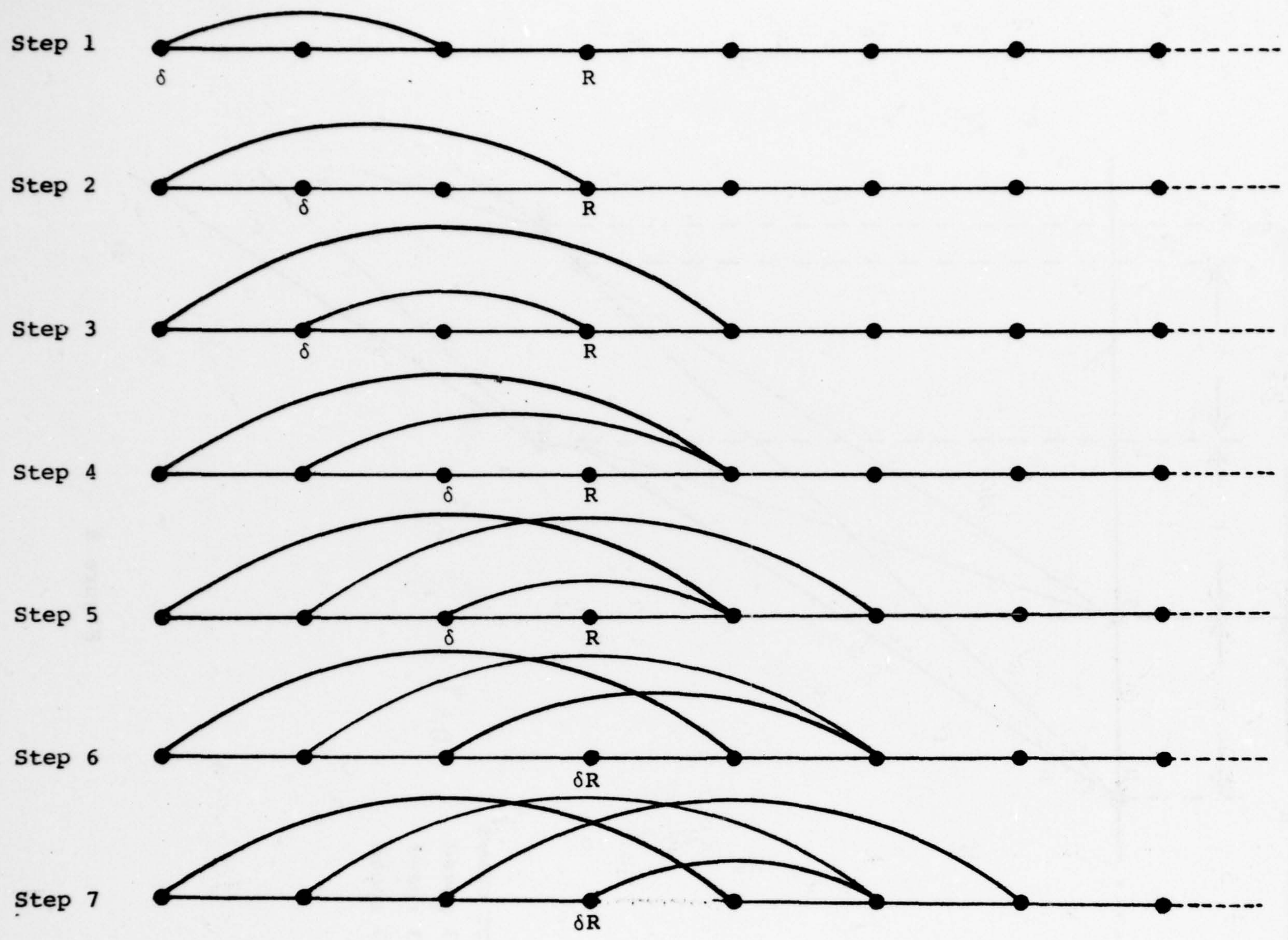
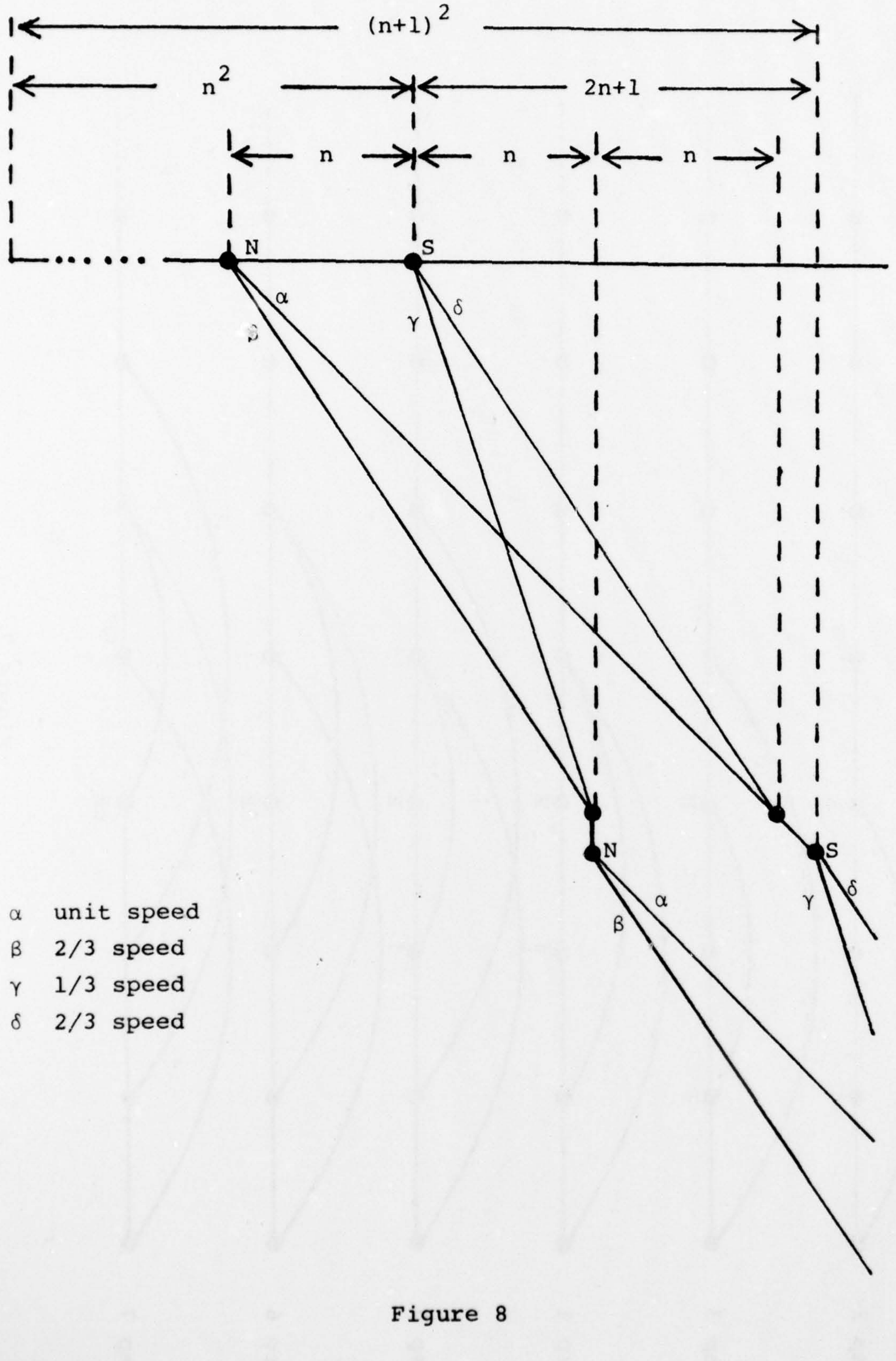


Figure 7



α	unit speed
β	2/3 speed
γ	1/3 speed
δ	2/3 speed

Figure 8

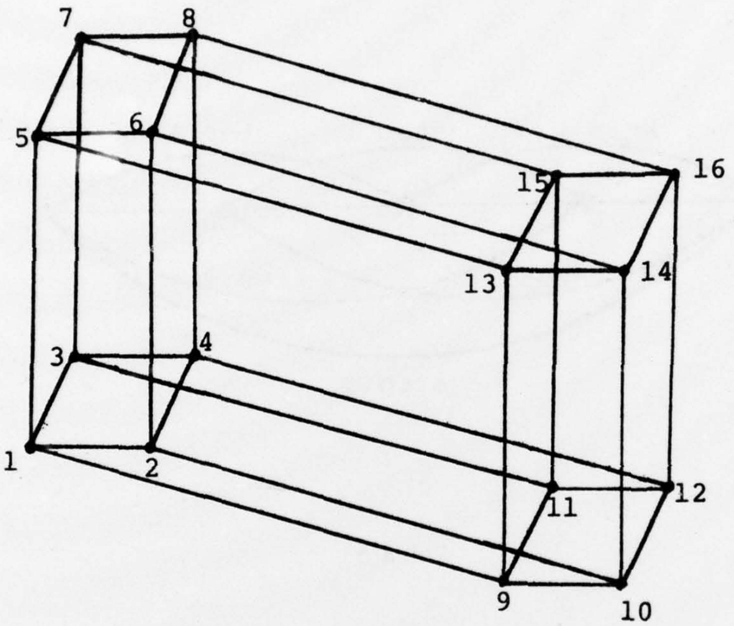


Figure 9

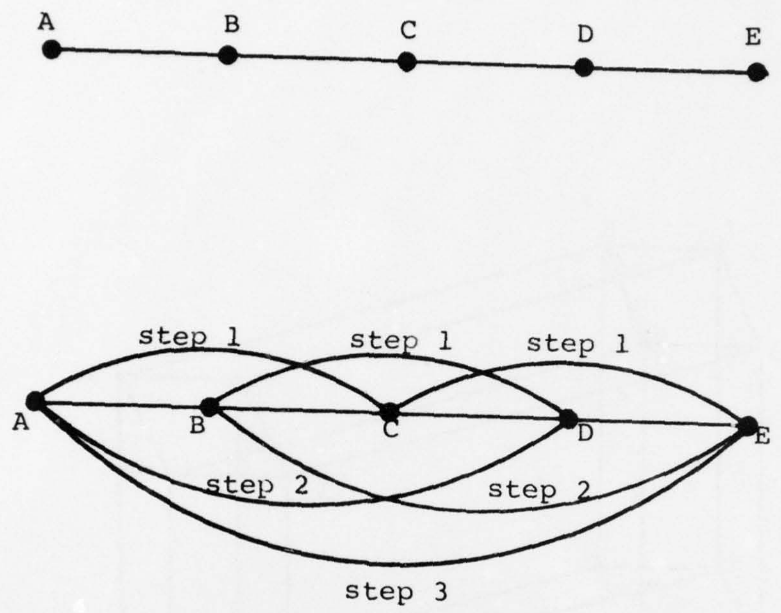


Figure 10

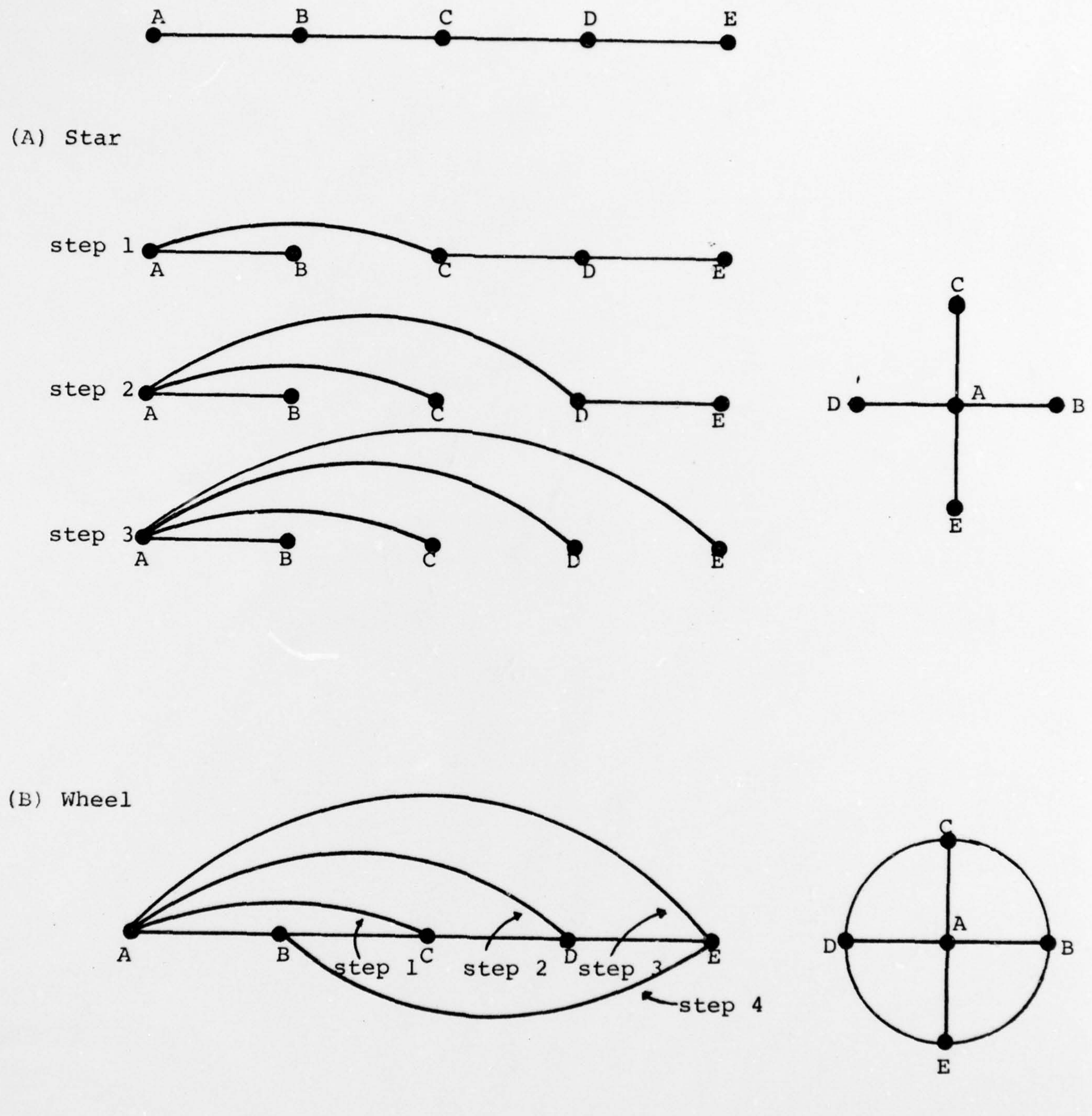

A
B
C
D
E
(A) Star
step 1
step 2
step 3
(B) Wheel
step 1
step 2
step 3
step 4

Figure 11

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) LOCAL RECONFIGURATION OF NETWORKS OF PROCESSORS		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER TR-730
7. AUTHOR(s) Angela Wu, Azriel Rosenfeld		8. CONTRACT OR GRANT NUMBER(s) AFOSR-77-3271
9. PERFORMING ORGANIZATION NAME AND ADDRESS Dept. of Mathematics, University of Md., Balto., MD 21228; Computer Science Center, Univ. of Md., College Park, MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Math. & Info. Sciences, AFOSR/NM Bolling AFB Wash., DC 20332		12. REPORT DATE February 1979
		13. NUMBER OF PAGES 40
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Networks
Graphs
Multiprocessing
Cellular computers
Reconfiguration

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper studies a "local" approach to reconfiguration of networks of processors, in which new connections are created only between pairs of processors that have a common neighbor. Algorithms for transforming strings into cycles, trees, arrays, hypercubes, cliques, stars and wheels, and vice versa are presented. The generation of all possible configurations of a given set of processors is also discussed.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)